CS10720 Problems and Solutions

Thomas Jansen

Today: Analysing Algorithms: Notation for Growth of Functions

February 18th

Organisational Issues

Important 'additional' lecture this Friday, 4-5pm, EL 0.26
(replacing the Monday lecture that fell victim to the storm)

Announcement I'll send out an email reminder about this tomorrow
because not everyone attends the lecture.
(But not everyone reads their emails, either. Ah, well...)

Important portfolio this week same contents as always
lecture summary Monday and Thursday
answer to this week's practicals questions
(not lecture summary Friday)
same deadline as always, Friday, 7pm

Portfolio feedback sent today

Plans for Today

Looking Back at Searching

Binary Search

```
int search(int *keys, int size, int key) {
  int left, right, middle;
  left=0;
  right=size-1;
  while ( left >= right) {
    middle = left + (right-left)/2;
    if ( keys[middle] == key )
      return middle;
    if (keys[middle] < key )
      left = middle+1;
    else
      right = middle-1;
  }
  return -1;
}
```

in the worst case

considers $\leq 2\log_2(n) + 2$ keys

Linear Search

```
int search(int *keys, int size, int key) {
  int i;
  i=0;
  while ( (i<size) && (keys[i]<key) )
    i++;
  if ( (i==size) || (keys[i]>key) )
    return -1;
  else
    return i;
}
```

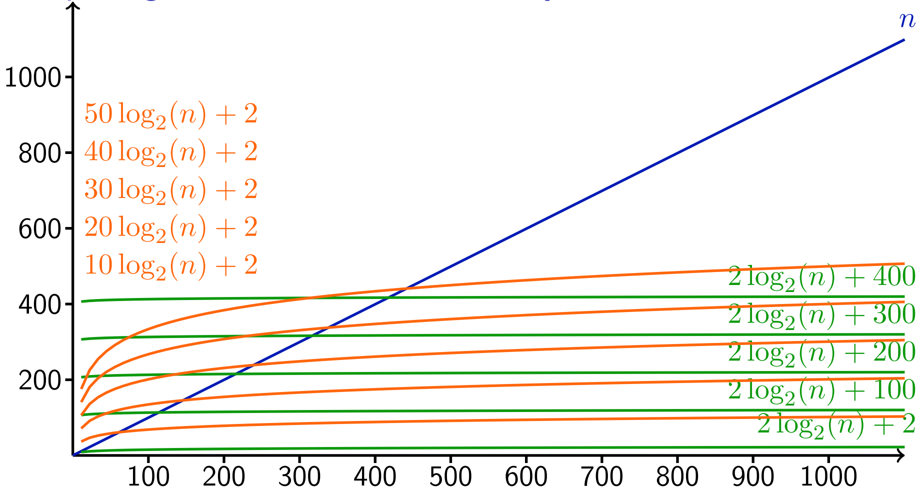
in the worst case

considers $\leq n$ keys ($n =$ size)

Does this really measure time? What about other operations?

Can we count so precisely for more complex algorithms?

Comparing Run Times: What Really Counts



Observation growth of 'main function' counts
not factor and not additive terms
at least for sufficiently large inputs

Measuring and Comparing Run Times

Remember growth of ‘main function’ counts
not factor and not additive terms
at least for sufficiently large inputs

Advantages

- things become a lot easier
  because we can count less precisely
- comparison of algorithms possible without knowing computer
  because we count less precisely
- no changes if we use a computer that is twice as fast
  because 2 is just a constant factor

Problem want to capture ‘growth of main function’ precisely
so that it is clear what we are doing

Downside requires some maths now

Upside saves us from a lot of maths later

Towards Comparing Functions

Remember want to capture 'growth of main function'
to be able to compare functions

How do we compare functions?

Idea start with something we already know
How do we compare numbers?

Remember when comparing two numbers a and b
there are five possibilities

1. $a \leq b$
2. $a \geq b$
3. $a = b$
4. $a < b$
5. $a > b$

Goal for Comparison of Functions

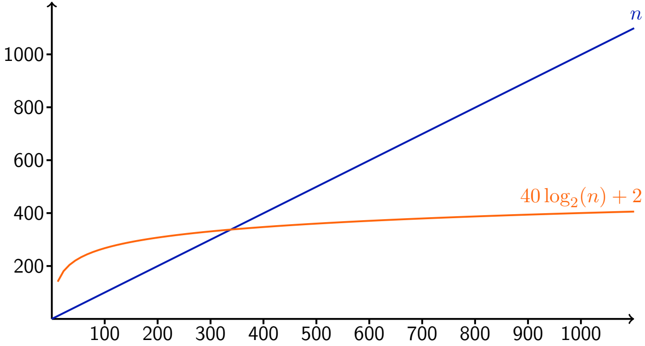
Remember want to capture 'growth of main function'
to be able to compare functions
similar to comparing numbers

numbers a and b	functions f and g	
$a \leq b$	'f grows at most as fast as g.'	$f = O(g)$
$a \geq b$	'f grows at least as fast as g.'	$f = \Omega(g)$
$a = b$	'f grows equally fast as g.'	$f = \Theta(g)$
$a < b$	'f grows slower than g.'	$f = o(g)$
$a > b$	'f grows faster than g.'	$f = \omega(g)$

Observation two 'real' definitions sufficient
because $\geq$ can be explained by 'reversing' $\leq$
$(a \geq b) \Leftrightarrow (b \leq a)$
$>$ can be explained by 'reversing $<$
$(a > b) \Leftrightarrow (b < a)$
$=$ can be explained by $\leq$ and $\geq$
$(a = b) \Leftrightarrow (a \leq b) \wedge (a \geq b)$

Defining 'Big O'

Remember



Definition (Big O Notation (f grows at most as fast as g))

Let $f, g\colon \mathbb{N}_0 \to \mathbb{R}^+$ be two functions. We say $f = O(g)$ if there exist constants $n_0 \in \mathbb{N}_0$ and $c \in \mathbb{R}^+$ such that for all $n \geq n_0$ the following holds: $f(n) \leq c \cdot g(n)$

About '$f = O(n)$'

Definition (Big O Notation (f grows at most as fast as g))

Let $f, g\colon \mathbb{N}_0 \to \mathbb{R}^+$ be two functions. We say $f = O(g)$ if there exist constants $n_0 \in \mathbb{N}_0$ and $c \in \mathbb{R}^+$ such that for all $n \geq n_0$ the following holds: $f(n) \leq c \cdot g(n)$

Observation The '$=$' in '$f = O(g)$' is **not an equal sign**!
(Well, it is, but it has not the usual meaning.)
It is not symmetric, i. e., $f = O(g) \not\Rightarrow g = O(f)$.

What is it if not an equal sign?

Well actually $O(g)$ is a set of functions
so in $f = O(g)$ the '$=$' is more like '$\in$'
we also allow '$O(f) = O(g)$', there the '$=$' is more like '$\subseteq$'

Why use '$=$' instead of '$\in$' and '$\subseteq$'?

Fact Graham/Knuth/Patashnik give four reasons
1 tradition, 2 tradition, 3 tradition, 4 convenience

Defining 'Big O', 'Big Omega' and 'Big Theta'

Definition ('Big O' (f grows at most as fast as g))

Let $f, g\colon \mathbb{N}_0 \to \mathbb{R}^+$ be two functions. We say $f = O(g)$
if there exist constants $n_0 \in \mathbb{N}_0$ and $c \in \mathbb{R}^+$
such that for all $n \geq n_0$ the following holds: $f(n) \leq c \cdot g(n)$

Definition ('Big Omega' (f grows at least as fast as g))

Let $f, g\colon \mathbb{N}_0 \to \mathbb{R}^+$ be two functions. We say $f = \Omega(g)$
if $g = O(f)$.

Definition ('Big Theta' (f grows equally fast as g))

Let $f, g\colon \mathbb{N}_0 \to \mathbb{R}^+$ be two functions. We say $f = \Theta(g)$
if $f = O(g)$ and $g = O(f)$.

Defining 'Little O' and 'Little Omega'

Definition ('Little O' (f grows slower than g))

Let $f, g\colon \mathbb{N}_0 \to \mathbb{R}^+$ be two functions. We say $f = o(g)$ if $\lim_{n\to\infty} f(n)/g(n) = 0$

Definition ('Little Omega' (f grows faster than g))

Let $f, g\colon \mathbb{N}_0 \to \mathbb{R}^+$ be two functions. We say $f = \omega(g)$ if $g = o(f)$.

Five definitions! How am I supposed to remember them all?

Observation remembering $O(\cdot)$ and $o(\cdot)$ suffices
because the others follow directly from those

But I don't know how to compute limits and maths is so hard!
How am I supposed to do this?
Don't panic! See rules of thumb (sufficient for CS107).

Rules of Thumb for Comparing Functions

- 'for polynomials the degree decides'
  $\forall c_1, c_2 \in \mathbb{R}^+ \colon (c_1 < c_2) \Rightarrow n^{c_1} = o(n^{c_2})$
- 'logarithm grows slower than any polynomial'
  $\forall c \in \mathbb{R}^+ \colon \log_2 n = o(n^c)$
- 'every polynomial grows slower than 2^n'
  $\forall c \in \mathbb{R}^+ \colon n^c = o(2^n)$
- 'every polynomial grows slower than any exponential function'
  $\forall b, c_1, c_2 \in \mathbb{R}^+, b > 1 \colon n^{c_1} = o(b^{c_2 \cdot n})$
- 'sum of functions is dominated by the largest function'
  $\forall f, g \colon \mathbb{N}_0 \to \mathbb{R}^+$ with $g = O(f) \colon f + g = \Theta(f)$

Looking Back at Searching

Binary Search

```
int search(int *keys, int size, int key) {
  int left, right, middle;
  left=0;
  right=size-1;
  while ( left >= right) {
    middle = left + (right-left)/2;
    if ( keys[middle] == key )
      return middle;
    if (keys[middle] < key )
      left = middle+1;
    else
      right = middle-1;
  }
  return -1;
}
```

in the worst case

considers $\leq 2\log_2(n) + 2$ keys

Linear Search

```
int search(int *keys, int size, int key) {
  int i;
  i=0;
  while ( (i<size) && (keys[i]<key) )
    i++;
  if ( (i==size) || (keys[i]>key) )
    return -1;
  else
    return i;
}
```

in the worst case

considers $\leq n$ keys ($n =$ size)

Observation

- Binary search has worst case run time $O(\log n)$.
- Linear search has worst case run time $O(n)$.

Fact

- Binary search has worst case run time $\Theta(\log n)$.
- Linear search has worst case run time $\Theta(n)$.

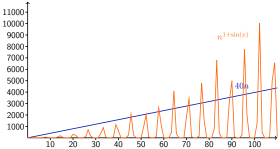
Some Examples

- $27n^2 = \Theta(n^2)$
  because we can ignore constant factors
- $14n^3 + 48n = \Theta(n^3)$
  because in sum largest function dominates
  and in polynomials the exponent decides
- $n \cdot (3n^2 + 12n + 18) = \Theta(n^3)$
  as above (after multiplication)
- $13n \cdot (2\log_2(n) + \log_2(\log_2(n))) = \Theta(n \log n)$
  because we can only ignore constant factors and $\log_2 n$ is not constant
- $\sqrt{n} + \log_2(n) = \Theta(\sqrt{n})$
  because $\log_2 n = o(n^c)$ for all positive c
- $\frac{1}{n} + \frac{12}{n^2} + 12 = \Theta(1)$
  because in the largest function dominates
  and 12 is ‘constant function’

Some General Questions, Answers and Examples

Can all functions $f, g\colon \mathbb{N}_0 \to \mathbb{R}^+$ be compared?

No! $f(n) = 40n$, $g(n) = n^{1+\sin(x)}$



Remember addition and multiplication 'easy' for O and Ω

Examples $O(n) + O(n^2) = O(n^2)$, $\Omega(n) \cdot \Omega(n) = \Omega(n^2)$

subtraction 'less so'

Examples $O(n) - O(n) = ?$, $\Omega(n^2) - \Omega(n) = ?$

Careful when adding a large number of functions

Example $O(n) + O(n) = O(n)$ ✔

$$\underbrace{O(n) + O(n) + \cdots + O(n)}_{\log_2(n) \text{ times}} = O(n \log n)$$

Summary & Take Home Message

Things to remember

- comparison of functions
- 'big O': f grows at most as fast as g: $f = O(g)$
- 'big Omega': f grows at least as fast as g: $f = \Omega(g)$
- 'big Theta': f grows equally fast as g: $f = \Theta(g)$
- 'little o': f grows slower than g: $g = o(g)$
- 'little omega': f grows faster than g: $g = \omega(g)$
- rules of thumb for the comparison of functions

Take Home Message

- Asymptotic notation simplifies the analysis of algorithms significantly.
- Asymptotic notation reveals true behaviour for sufficiently large input sizes.
- For small input sizes a closer look is sometimes required.

Lecture feedback `http://onlineted.com`